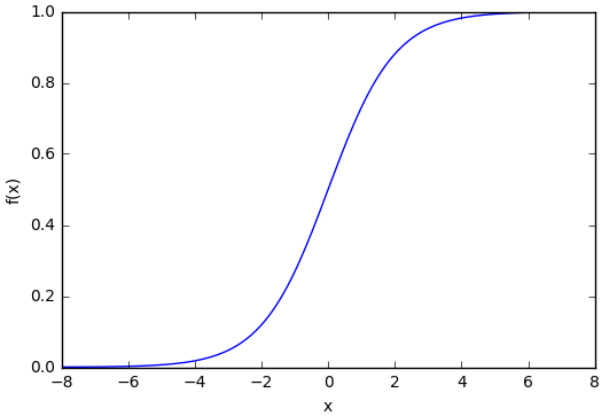
Neural network

29 апреля 2019 г.

В качестве активационной функции обычно используют сигмоидальную функцию:

$$f(z) = \frac{1}{1 + exp(-z)}$$



Для каждого узла считается значение:

$$x_1(w)_1 + x_2(w)_2 + x_3(w)_3 + b$$

Процесс прямого распространения

$$h_1^{(2)} = f\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right)$$
$$h_2^{(2)} = f\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}\right)$$
$$h_3^{(2)} = f\left(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}\right)$$

где $f()$ — активационная функция узла, в нашем случае сигмоидальная функция. В первой строке $h_1^{(2)}$— выход первого узла во втором слое, его входами соответственно являются $w_{11}^{(1)}x_1^{(1)}, w_{12}^{(1)}x_2^{(1)}, w_{13}^{(1)}x_3^{(1)}$ и $b_1^{(1)}$. Эти входы было сложены, а затем переданы в активационную функцию для расчета выхода первого узла. С двумя следующими узлами аналогично.

Это эквивалентно перемножению мвтриц:

$$f\left(\begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \\ w_{31}^{(1)} & w_{32}^{(1)} & w_{33}^{(1)} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{bmatrix}\right)$$

Градиентный спуск и оптимизация

$$w_{new} = w_{old} - \alpha \nabla error$$

Функция оценки:

$$\begin{aligned} J(w,b) &= \frac{1}{m}\sum_{z=0}^{m}\frac{1}{2}\parallel y^{z} - h^{(n_l)}(x^{z})\parallel^2 \\ &= \frac{1}{m}\sum_{z=0}^{m} J(W,b,x^{(z)},y^{(z)}) \end{aligned}$$

$$h_{W,b}(x) = h_1^{(3)} = f(w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)})$$

Мы можем упростить это уравнение к $h_1(3)=f(z_1^{(2)})$, добавив новое значение $z_1^{(2)}$, которое означает:

$$z_1^{(2)} = w_{11}^{(2)}h_1^{(2)} + w_{12}^{(2)}h_2^{(2)} + w_{13}^{(2)}h_3^{(2)} + b_1^{(2)}$$

Предположим, что мы хотим узнать, как влияет изменение в весе $w_{12}^{(2)}$ на функцию оценки. Это означает, что нам нужно вычислить $\partial J/\partial w_{12}^{(2)}$. Чтобы сделать это, нужно использовать правило дифференцирования сложной функции:

$$\frac{\partial J}{\partial w_{12}^{(2)}} = \frac{\partial J}{\partial h_1^{(3)}}\frac{\partial h_1^{(3)}}{\partial z_1^{(2)}}\frac{\partial z_1^{(2)}}{\partial w_{12}^{(2)}}$$

Применяя эту идею мы можем получить распространение ошибки в скрытых слоях:

$$w_{ij}^{(l)} = w_{ij}^{(l)} - \alpha \frac{\partial}{\partial w_{ij}^{(l)}} J(w,b)$$

$$b_{i}^{(l)} = b_{i}^{(l)} - \alpha \frac{\partial}{\partial b_{i}^{(l)}} J(w,b)$$

И, наконец, мы пришли к определению метода обратного распространения через градиентный спуск для обучения наших нейронных сетей. Финальный алгоритм обратного распространения выглядит следующим образом:
Рандомная инициализация веса для каждого слоя $W^{(l)}$. Когда итерация < границы итерации:

01. Зададим ΔW и Δb начальное значение ноль.
02. Для экземпляров от 1 до m: a. Запустите процесс прямого распространения через все n_l слоев. Храните вывод активационной функции в $h^{(l)}$б. Найдите значение $\delta^{(nl)}$ выходного слоя. Обновите $\Delta W^{(l)}$и $\Delta b^{(l)}$ для каждого слоя.
03. Запустите процесс градиентного спуска, используя:

$$W^{(l)} = W^{(l)} - \alpha \left[\frac{1}{m} \Delta W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

```
In [1]: import pandas
        import cv2 as cv
        import cv2
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score
        import numpy.random as npr
```

```
In [2]: def resize(image, width = 64, height = 64):
            dim = (width, height)
            resized = cv2.resize(image, dim, interpolation = cv2.INTER_AREA)
            return resized
```

```
        def bgr2rgb(img):
            b,g,r = cv.split(img)
            return cv.merge([r,g,b])

        def rgb2bgr(img):
            b,g,r = cv.split(img)
            return cv.merge([r,g,b])

        def get_grey(img):
            w, h, d = img.shape
            img = img.reshape((w*h), d)
            img_new = (0.2989 * img[:,0] + 0.5870 * img[:,1] + 0.1140 * img[:,2])
            img_new = img_new.round().astype(int)
            return img_new.reshape((w,h))

        def convert_y_to_vect(y, nn_structure):
            y_vect = np.zeros((len(y), nn_structure[2]))
            for i in range(len(y)):
                y_vect[i, y[i]] = 1
            return y_vect
```

In [3]:
```
        def f(x):
            return 1 / (1 + np.exp(-x))

        def f_deriv(x):
            return f(x) * (1 - f(x))
```

In [4]:
```
        def setup_and_init_weights(nn_structure):
            W = {}
            b = {}
            for l in range(1, len(nn_structure)):
                W[l] = npr.random_sample((nn_structure[l], nn_structure[l-1]))/1000
                b[l] = npr.random_sample((nn_structure[l],))/1000
            return W, b

        def init_tri_values(nn_structure):
            tri_W = {}
            tri_b = {}
            for l in range(1, len(nn_structure)):
                tri_W[l] = np.zeros((nn_structure[l], nn_structure[l-1]))
                tri_b[l] = np.zeros((nn_structure[l],))
            return tri_W, tri_b
```

In [5]:
```
        def feed_forward(x, W, b):
            h = {1: x}
```

```
            z = {}
            for l in range(1, len(W) + 1):
                if l == 1:
                    node_in = x
                else:
                    node_in = h[l]
                z[l+1] = W[l].dot(node_in) + b[l]
                h[l+1] = f(z[l+1])
            return h, z
```

In [6]:
```
def calculate_out_layer_delta(y, h_out, z_out):
    return -(y-h_out) * f_deriv(z_out)

def calculate_hidden_delta(delta_plus_1, w_l, z_l):
    return np.dot(np.transpose(w_l), delta_plus_1) * f_deriv(z_l)
```

In [7]:
```
def train_nn(nn_structure, X, y, iter_num=300, alpha=0.25):
    W, b = setup_and_init_weights(nn_structure)
    cnt = 0
    m = len(y)
    avg_cost_func = []
    print('Starting gradient descent for ', iter_num, ' iterations')
    while cnt < iter_num:
        tri_W, tri_b = init_tri_values(nn_structure)
        avg_cost = 0
        for i in range(len(y)):
            delta = {}
            h, z = feed_forward(X[i, :], W, b)
            for l in range(len(nn_structure), 0, -1):
                if l == len(nn_structure):
                    delta[l] = calculate_out_layer_delta(y[i,:], h[l], z[l])
                    avg_cost += np.linalg.norm((y[i,:]-h[l]))
                else:
                    if l > 1:
                        delta[l] = calculate_hidden_delta(delta[l+1], W[l], z[l]
                    tri_W[l] += np.dot(delta[l+1][:,np.newaxis],
                                       np.transpose(h[l][:,np.newaxis]))
                    tri_b[l] += delta[l+1]
        for l in range(len(nn_structure) - 1, 0, -1):
            W[l] += -alpha * (1.0/m * tri_W[l])
            b[l] += -alpha * (1.0/m * tri_b[l])
        avg_cost = 1.0/m * avg_cost
        avg_cost_func.append(avg_cost)
        cnt += 1
    return W, b, avg_cost_func
```

In [8]:
```
def predict_y(W, b, X, n_layers):
    m = X.shape[0]
    y = np.zeros((m,))
    for i in range(m):
        h, z = feed_forward(X[i, :], W, b)
        y[i] = np.argmax(h[n_layers])
    return y

def show_work(W, b, X, n_layers, i):
    plt.imshow(X_test[i].reshape((8,8)), cmap="gray")
    print(predict_y(W, b, X_test, 3)[i])
```

In [9]:
```
data = pandas.read_csv(
'E:\\User\\Desktop\\Khlamskaya_prog\\IAD\\img_align_celeba\\list_attr_celeba.csv

df = pandas.DataFrame(data.values[1:], columns=data.values[0])
sourse = df[['img']].values
target = df[['Male']].values

sourse = sourse.reshape(sourse.shape[0])
target = target.reshape(target.shape[0])

for i in df.index :
    if(target[i] == '1'):
        target[i] = 1
    else :
        target[i] = 0

def prep_sourse(sourse, num = 20, start = 0, w = 64, h = 64):
    X_train = np.zeros((num, w*h))
    for i in range(num):
        name = sourse[start + i]
        img = bgr2rgb(cv.imread(
        ' E:\\User\\Desktop\\Khlamskaya_prog\\IAD\\img_align_celeba\\images\\' +
        img = resize(img, w, h)
        img = get_grey(img).reshape(-1)
        X_train[i] = img
    return X_train

p_resized_shape = 32
num = 300
start = 100

nn_structure = [p_resized_shape*p_resized_shape, 1000, 2]
```

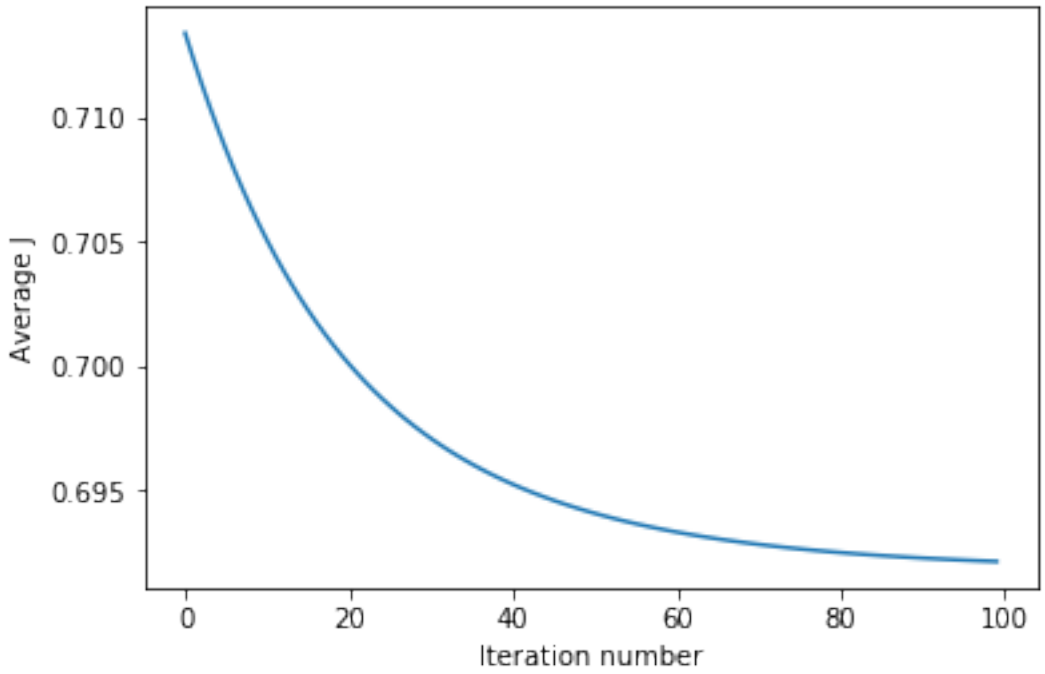
```
X = prep_sourse(sourse, num = num, start = start,
                w = p_resized_shape, h = p_resized_shape)
X = X/255
target = target[start: start + num]

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    target, test_size=0.4)
y_v_train = convert_y_to_vect(y_train, nn_structure)
y_v_test = convert_y_to_vect(y_test, nn_structure)

W, b, avg_cost_func = train_nn(nn_structure, X_train,
                               y_v_train, iter_num=100, alpha=0.002)
plt.plot(avg_cost_func)
plt.ylabel('Average J')
plt.xlabel('Iteration number')
plt.show()

y_pred = predict_y(W, b, X_test, 3)
y_test = np.array(y_test, dtype=int)
print('Prediction accuracy is', accuracy_score(y_test, y_pred) * 100,'%')
```

```
Starting gradient descent for  100  iterations
```



```
Prediction accuracy is 66.67 %
```